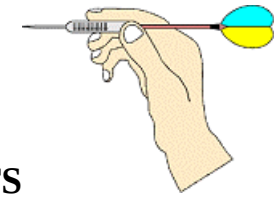
Automated Software Testing - A Perspective

Note from the author:
My perspective on most things is that the 'glass is half full' rather than half empty. This attitude carries over to the advice I suggest on automated software testing as well. I should point out, however, there is an ever increasing awareness from others experienced in this field, as well as from my own experience, that many efforts in test automation do not live up to expectations. A lot of effort goes into developing and maintaining test automation, and even once it's built you may or may not recoup your investment. It's very important to perform a good cost/benefit analysis on whatever manual testing you plan to automate. The successes I've seen have mostly been on focused areas of the application where it made sense to automate, rather than complete automation efforts. Also, skilled people were involved in these efforts and they were allowed the time to do it right.

Test automation can add a lot of complexity and cost to a test team's effort, but it can also provide some valuable assistance if its done by the right people, with the right environment and done where it makes sense to do so. I hope by sharing some pointers that I feel are important that you'll find some value that translates into saved time, money and less frustration in your efforts to implement test automation back on the job.
Kerry



KEY POINTS

I've listed the 'key points' up front instead of waiting until the end. The rest of the article will add detail to some of these key points.

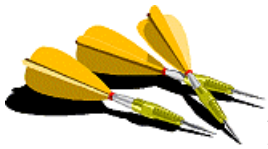
- First, it's important to define the purpose of taking on a test automation effort. There are several categories of testing tools each with its own purpose. Identifying what you want to automate and where in the testing life cycle will be the first step in developing a test automation strategy. Just wishing that everything should be tested faster is not a practical strategy. You need to be specific.

- Developing a test automation strategy is very important in mapping out what's to be automated, how it's going to be done, how the scripts will be maintained and what the expected costs and benefits will be. Just like every testing effort should have a testing strategy, or test plan, so should there be a 'plan' built for test automation.

- Many of the testing 'tools' provided by vendors are very sophisticated and use existing or proprietary coding 'languages'. The effort of automating an existing manual testing effort is no different than a programmer using a coding language to write programs to automate any other manual process. Treat the entire process of automating testing as you would any other software development effort. This includes defining what should be automated, (the requirements phase), designing test automation, writing the scripts, testing the scripts, etc. The scripts need to be maintained over the life of the product just as any program would require maintenance. Other components of software development, such as configuration management also apply.

- The effort of test automation is an investment. More time and resources are needed up front in order to obtain the benefits later on. Sure, some scripts can be created which will provide immediate payoff, but these opportunities are usually small in number relative to the effort of automating most test cases. What this implies is that there usually is not a positive payoff for automating the current release of the application. The benefit comes from running these automated tests every subsequent release. Therefore, ensuring that the scripts can be easily maintained becomes very important.

- Since test automation really is another software development effort, it's important that those performing the work have the correct skill sets. A good tester does not necessarily make a good test automator. In fact, the job requirements are quite different. Good testers are still necessary to identify and write test cases for what needs to be tested. A test automator, on the other hand, takes these test cases and writes code to automate the process of executing those tests. From what I've seen, the best test automation efforts have been lead by developers who have put their energies into test automation. That's not to say that testers can't learn to be test automators and be successful, it's just that those two roles are different and the skill sets are different.

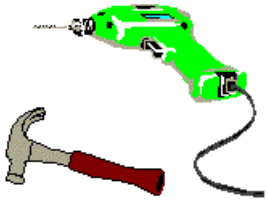


POINTS

Here are some other important points to consider.

- When strategizing for test automation, plan to achieve small successes and grow. It's better to incur a small investment and see what the effort really takes before going gung ho and trying to automate the whole regression suite. This also gives those doing the work the opportunity to try things, make mistakes and design even better approaches.

- Many software development efforts are underestimated, sometimes grossly underestimated. This applies to test automation as well, especially if the effort is not looked upon as software development. Test automation is not something that can be done on the side and care should be taken when estimating the amount of effort involved. Again, by starting small and growing, estimating the work can be gauged.

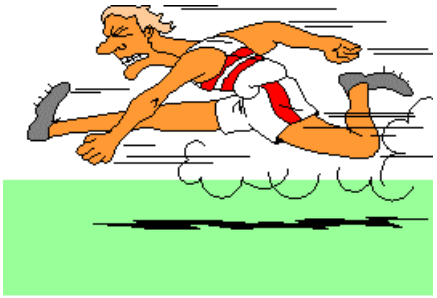


- When people think of testing tools, many first think of the 'capture/playback' variety where the application is tested at the end during system test. There are several types of testing tools which can be applied at various points of code integration. Test automation can be applied at each of the levels of testing including unit testing, one or more layers of integration testing, and system testing (another form of integration). The sooner tests can be executed after the code is written, before too much code integration has occurred, the more likely bugs will not be carried forward. When strategizing for test automation, consider automating these tests as early as possible as well as later in the testing life cycle.

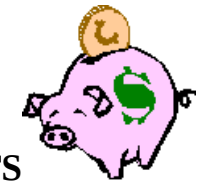
- Related to this last point is the idea that testers and software developers need to work as a team to make effective test automation work. I don't believe testing independence is lost when testers and developers work together, but there can be some excellent advantages which I'll later point out.

- Testing tools, as sophisticated as they have become, are still dependent upon consistency in the test environment. This should be quite obvious, but having a dedicated test environment is absolutely necessary. If testers don't have control of their test environment and test data, the required setup for tests may not meet the requirements of those tests. When manual testing is done testers may sometimes 'work around' test setup issues. Automated test scripts are less flexible requiring specific setup scenarios thereby needing more control.

- Test automation is not the only answer to delivering quality software. In fact, test automation in many cases is a last gasp effort in an attempt to find problems after they've been made instead of eliminating the problems as they are being created. Test automation is not a substitute for walkthroughs, inspections, good project management, coding standards, good configuration management, etc. Most of these efforts produce a higher pay back for the investment than does test automation. Testing will always need to be done and test automation can assist, but it should not be looked upon as the primary activity in producing better software.



The truth is that developers can produce code faster and faster with more complexity than ever before. Advancements in code generation tools and code reuse are making it difficult for testers to keep up with software development. Test automation, especially if applied only at the end of the testing cycle, will not be able to keep up with these advances. We must pull out all stops along the development life cycle to build in good quality software and test as early and often as possible with the assistance of test automation.



BENEFITS

To many people, the benefits of automation are pretty obvious. Tests can be run faster, they're consistent, and tests can be run over and over again with less overhead. As more automated tests are added to the test suite more tests can be run each time thereafter. Manual testing never goes away, but these efforts can now be focused on more rigorous tests.

There are some common 'perceived' benefits that I like to call 'bogus' benefits. Since test automation is an investment it is rare that the testing effort will take less time or resources in the current release. Sometimes there's the perception that automation is easier than testing manually. It actually makes the effort more complex since there's now another added software development effort. Automated testing does not replace good test planning, writing of test cases or much of the manual testing effort.

COSTS

Costs of test automation include personnel to support test automation for the long term. As mentioned, there should be a dedicated test environment as well as the costs for the purchase, development and maintenance of tools. All of the efforts to support software development, such as planning, designing, configuration management, etc. apply to test automation as well.

COMMON VIEW

Now that some of the basic points have been noted, I'd like to talk about the paradigm of testing automation. When people think of test automation, the 'capture/playback' paradigm is commonly perceived. The developers create the application software and turn it over to the testing group. The testers then busily use capture/playback functionality of the testing tool to quickly create test scripts. Capture/playback is used because it's easier than 'coding' scripts. These scripts are then used to test the application software.

There are some inherent problems with this paradigm. First, test automation is only applied at the final stage of testing when it is most expensive to go back and correct the problem. The testers don't get a chance to create scripts until the product is finished and turned over. At this point there is a tremendous pull on resources to just test the software and forgo the test automation effort. Just using capture/playback may be temporarily effective, but using capture/playback to create an entire suite will make the scripts hard to maintain as application modifications are made.

TEST and AUTOMATE EARLY

From observations and experience, a different paradigm appears to be more effective. Just as you would want to test early and test often if you were testing manually, the same applies to test automation. The first level of testing is the unit testing performed by the developer. From my experience unit testing can be done well or not done well depending on the habits and personality of the developer. Inherently, developers like to develop, not write test cases. Here's where an opportunity for developers and testers to work together can begin to pay off. Testers can help document unit tests and developers can write utilities to begin to automate their unit tests. Assisting in documenting test cases will give a better measurement of unit tests executed. Much success of test automation comes from homegrown utilities. This is because they integrate so well with the application and there is support from the developer to maintain the utilities so that they work with the application. More effective and efficient unit testing, through the use of some automation, provides a significant bang for the buck in trying to find bugs in the testing life cycle. Static analyzers can also be used to identify which modules have the most code complexity and may require more testing.

WORK WITH DEVELOPERS

The same approach should be applied at each subsequent level of testing. Apply test automation where it makes sense to do so. Whether homegrown utilities are used or purchased testing tools, it's important that the development team work with the testing team to identify areas where test automation makes sense and to support the long term use of test scripts.

Where GUI applications are involved the development team may decide to use custom controls to add functionality and make their applications easier to use. It's important to determine if the testing tools used can recognize and work with these custom controls. If the testing tools can't work with these controls, then test automation may not be possible for that part of the application. Similarly, if months and months of effort went into building test scripts and the development team decides to use new custom controls which don't work with existing test scripts, this change may completely invalidate all the effort that went into test automation. In either case, by identifying up front in the

application design phase how application changes affect test automation, informed decisions can be made which affect application functionality, product quality and time to market. If test automation concerns aren't addressed early and test scripts cannot be run, there is a much higher risk of reduced product quality and increased time to market.

Working with developers also promotes building in 'testability' into the application code. By providing hooks into the application testing can sometimes be made more specific to any area of code. Also, some tests can be performed which otherwise could not be performed if these hooks were not built.

Besides test drivers and capture/playback tools, code coverage tools can help identify where there are holes in testing the code. Remember that code coverage may tell you if paths are being tested, but complete code coverage does not indicate that the application has been exhaustively tested. For example, it will not tell you what has been 'left out' of the application.

CAPTURE/PLAYBACK

Here's just a note on capture/replay. People should not expect to install the testing tool, turn on the capture function and begin recording tests which will be used forever and ever. Capturing keystrokes and validating data captured within the script will make the script hard to maintain. Higher level scripts should be designed to be modular which has options to run several tests scripts. The lower level test scripts that actually perform tests also should be relatively small and modular so they can be shared and easily maintained. Data for input should not be hard coded into the script, but rather read from a file or spreadsheet and loop through the module for as many times as you wish to test with variations of data. The expected results should also reside in a file or spreadsheet and read in at the time of verification. This method considerably shortens the test script making it easier to maintain and possibly reuse by other test scripts. Bitmap comparisons should be used very sparingly. The problem with bitmap comparison is that if even one pixel changes in the application for the bitmap being compared, the image will compare as a mismatch even if you recognize it as a desirable change and not a bug. Again, the issue is maintainability of the test suite.

Capture/playback functionality can be useful in some ways. Even when creating small modular scripts it may be easier to first capture the test then go back and shorten and modify it for easier maintenance. If you wish to create scripts which will obviously provide immediate pay back, but you don't care if it's maintainable, then using capture/playback can be a very quick way to create the automated test. These scripts typically are thrown away and rebuilt later for long term use. The capture/playback functionality is also good to use during the design phase of a product if there's a prototype developed. During usability testing, which is an application design technique, users sit at the computer using a mock up of the actual application where they're able to use the interface, but the real functionality has not yet been built. By running the capture/playback tool in capture mode while the users are 'playing' with the application, recorded keystrokes and mouse movements can track where the users move on the system. Reading these captured scripts help the designers understand the level of difficulty in navigating through the application.

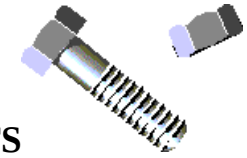
PLAYERS

Test automation is not just the responsibility of the testers. As noted, getting developers involved is important as well as getting the understanding and support of management. Since test automation is an investment, it's important that they understand the up front costs and expected benefits so that test automation stays around long enough to show the benefits. There is the tendency to 'give up' when results are not shown right away.

If the project is just beginning with test automation then having someone who can champion the test automation effort is important. This 'champion' should have skills in project management, software testing and software development (preferably a coding background). This 'champion' is responsible for being the project manager of the test automation effort. This person needs to interact well with both the testers and the application developers. Since this person may also be actively involved with writing scripts as well, good development skills are also desirable. This person should not be involved with the designing of test cases or manual testing other than to review other team member's work. Typically there is not enough time to both design test cases and design test automation. Nor is there time to build test scripts and run manual tests by the same person. Where the testing effort is large the distinction between these two roles apply to teams of automators and testers as well. Too many times test automators are borrowed to performed manual testing never to realize the benefits of test automation in the current or future releases of the application.

This is not to say that the role of testers is reduced. Test planning still needs to be done by a test lead, test cases still need to be designed and manual testing will still be performed. The added role for these testers is that they most likely will begin to run the automated test scripts. As they run these scripts and begin to work more closely with the test automation 'champion' or test automators, they too can begin to create scripts as the automated test suite matures.

Experience has shown that most bugs are not found by running automated tests. Most bugs are found in the process of creating the scripts, or the first time the code is tested. What test automation mostly buys you is the opportunity to not spend valuable man-hours re-testing code that has been tested before, but which has to be tested in any case because the risk is too high not to test it. The other benefit comes from the opportunity to spend these man-hours rigorously testing new code for the first time and identifying new bugs. Just as testing in general is not a guarantee, but a form of insurance, test automation is a method to have even more insurance.



SOME NUTS AND BOLTS

When learning to use testing tools it's common to make mistakes. One way to mitigate these mistakes is to create scripts that will provide immediate pay back. That is, create scripts which won't take too much time to create yet will obviously save manual testing effort. These scripts will be immediately useful and it's all right if they're not intended to be part of the permanent test suite. Those creating the scripts will learn more about the tool's functionality and learn to design even better scripts. Not much is lost if these scripts are thrown away since some value has already been gained from them. As experience is gained with the testing tool, a long term approach to test automation design can start to be developed.

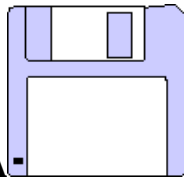
Again, start off small when designing scripts. Identify the functional areas within the application being tested. Design at a higher level how each of these functional areas would be automated, then create a specific automated test design for one of the functional areas. That is, what approach will be used to create scripts using test cases as the basis for automating that function? If there are opportunities to use common scripting techniques with other testing modules, then identifying these common approaches as potential standards would be useful in creating maintainable scripts.

Use a similar approach to design and create scripts for some of the other functional areas of the application. As more experience is gained from automation then designing and building scripts to test the integration of these functional areas would be the next step in building a larger and more useful testing suite.

Since the purpose of automating testing is to find bugs, validations should be made as tests are performed. At each validation point there is a possibility of error. Should the script find an error, logic should be built into it so that it can not only report the error it found but also route back to an appropriate point within the automated testing suite so that the automated testing can continue on. This is necessary if automated tests are to be run overnight successfully. This part of test automation is the 'error recovery process'. This is a significant effort since it has to be designed in for every validation point. It's best to design and create reusable error recovery modules that can be called from many validation points in many scripts. Related to this are the reports that get generated from running the tests. Most tools allow you to customize the reports to fit your reporting needs.

It's also important to write documented comments in the test scripts to help those who would maintain the test scripts. Write the scripts with the belief that someone else will be maintaining them.

In the automation test design or documented within the test scripts also identify any manual intervention that is necessary to set up the test environment or test data in order to run the scripts. Perhaps databases need to be loaded or data has to be reset.



TEST DATA-

I know of three ways to have the test data populated so that the test environment is setup correctly to run automated tests. If complete control of the test environment is available to testers, then reloading preset databases can be a relatively quick way to load lots of data. One danger in having several preset databases is if a future release requires a reconstruction of data structures and the effort to convert the current data structures to the desired state is a large effort.

Another method of setting up the data is to create tests scripts that run and populate the database with the necessary data to be used in automated tests. This may take a little longer to populate, but there's less dependency on data structures. This method also allows more flexibility should other data change in the database.

Even though I mention 'databases' specifically, the concepts apply to other types of data storage as well.

Other people with test automation experience have used 'randomly' generated data successfully to work with their test scripts. Personally, I have no experience using randomly generated data, but this is another option worth looking into if you're looking for other ways to work with data.

POTENTIAL RISKS

Some common risks to the test automation effort include management and team members support fading after not seeing immediate results, especially when resources are needed to test the current release. Demanding schedules will put pressure on the test team, project management and funding management to do what it takes to get the latest release out. The reality is that the next release usually has the same constraints and you'll wish you had the automated testing in place.

If contractors are used to help build or champion the test automation effort because of their experience, there is the risk that much of the experience and skills will 'walk away' when the contractor leaves. If a contractor is used, ensure there is a plan to back fill this position since the loss of a resource most likely will affect the maintenance effort and new development of test scripts. It's also just as important that there is a comprehensive transfer of knowledge to those who will be creating and maintaining the scripts.

Since the most significant pay back for running automated tests come from future releases, consider how long the application being tested will remain in its current state. If a rewrite of the application is planned in the near future or if the interface is going to be overhauled, then it probably makes sense to only use test automation for immediate pay back. Again, here's where working with application designers and developers can make a difference, especially if internal changes are planned that may not appear to affect the testing team, but in reality can affect a large number of test scripts.

SUMMARY

As mentioned earlier, most of the concepts identified here came from experiences and as also noted there's not a lot of facts to back up these ideas. The intent here wasn't to prove any particular technique worked, but, rather just to share methods that appear to be more successful. If nothing else, this information can be used to look at test automation from a little different perspective and assist in planning.

If you have experiences that are different than these that you've found successful, or if you've experienced hardships using some of these recommendations, I'd be grateful to hear from you. Many people, including myself, are interested in finding out what really works in creating higher quality software more quickly.

Comments can be sent to me at: zallar@testingstuff.com

I also support a testing web page that has links to quality organizations, tool vendors, consultants, and other testing references. It is not the best testing web page out there, but there are pointers to some other excellent web pages and they are noted. This URL for this page is: http://www.testingstuff.com